CHAPTER

4

RUNNING A LINUX SYSTEM

After reading this chapter and completing the exercises, you will be able to:

- Work with files and directories on a Linux system using basic commands
- Run programs and manage the corresponding software packages using popular data formats
- Add or remove features from the Linux kernel
- Review and change the initialization process that starts a Linux-based computer

In the previous chapter you learned about installing a new Linux system from a CD. You learned how to set up Linux hard disk partitions, how to answer configuration questions, and how to log in to a newly installed Linux system.

In this chapter you will learn about how to work with the files and directories on a Linux system and manage the software packages installed on the system. You will also learn about internal Linux operations, including the basics of the Linux kernel and the initialization process.

WORKING WITH LINUX FILES AND DIRECTORIES

From your work with computers, you know that manipulating files and directories is a large part of interacting with a computer. You search directories for a certain file, you open a file to review or change its contents, you save an updated version of a file, and so forth. Linux operates in much the same way as other operating systems: your data is stored in files; files are arranged in directories. Multiple directories are arranged in a branching treelike structure, in which one directory leads to other directories. Directories are also commonly represented by folders that can contain other folders (that is, other directories) and documents (or files). The directories contained within another directory are called subdirectories. Each subdirectory can in turn contain its own subdirectories.

Within Linux, every data file and device is part of one large directory structure. All directories in the Linux directory structure branch out from the root directory. The **root directory**, represented by a single forward slash (/), is the starting point for all access to Linux resources. All Linux configuration files are located in subdirectories of the root directory. All devices are also associated with a file located in a subdirectory of the root directory. (You may remember examples such as `/dev/hda1` from previous chapters.) As a system administrator, much of the work that you do to maintain a Linux system consists of reviewing the contents of files, updating files, and otherwise managing the status, contents, and location of files and the directories that contain them.

Table 4-1 Standard Linux Subdirectories of the Root Directory

Directory	Contents
`/etc`	Configuration files.

By default, Linux includes a standard set of subdirectories that branch off the root directory. Table 4-1 lists these subdirectories and what they contain. Some versions of Linux may contain subdirectories of the root directory that are not listed here, but Table 4-1 contains those that are found on all Linux systems. You will become very familiar with all of these main subdirectories as you learn more about Linux.

Table 4-1 Standard Linux Subdirectories of the Root Directory

Directory	Contents
`/etc`	Configuration files.
`/var`	Variable (changing) information, including system log files (see Chapter 11), e-mail messages, and files being printed.
`/home`	Home directories for all regular user accounts.
`/bin`	Executable programs.
`/sbin`	Executable programs used only by the `root` user.
`/usr`	Files (data, programs, documentation) used by all regular users on the Linux system. This subdirectory contains many other files and subdirectories.
`/tmp`	Temporary files used by the system or regular users.
`/root`	Home directory of the `root` user (the `superuser` account).
`/boot`	Files used to initialize Linux when the system is booted.
`/dev`	Files used to access hardware resources (devices) on the computer system.
`/lib`	System libraries (described later in this chapter) used by many Linux programs.

On most Linux systems, the largest of the directories in Table 4-1 (that is, the one containing the most files and subdirectories), is the `/usr` subdirectory. This subdirectory contains nearly all of the programs you regularly use in Linux (such as the commands described in the following sections), the files for the graphical system, all the standard Linux online documentation, and many other types of files.

The names and arrangement of the Linux subdirectories may not be intuitive—the names are short and strange. But each one has a distinct purpose that has been refined over the years of UNIX and Linux development. This predefined directory arrangement ensures that a program can locate the files it needs (such as system configuration files) on any Linux system. The best way to become familiar with the arrangement and purpose of the hundreds of files included with a standard Linux distribution is to use the `cd` and `ls` commands (explained in "Linux File Commands") to explore your system's directory structure.

Working at a Command Line

After you log in (using a valid username and password), the system may be in a text-mode (also called character-mode or character-cell) environment, in which case you see a prompt where you can type commands. This environment is shown in Figure 4-1.

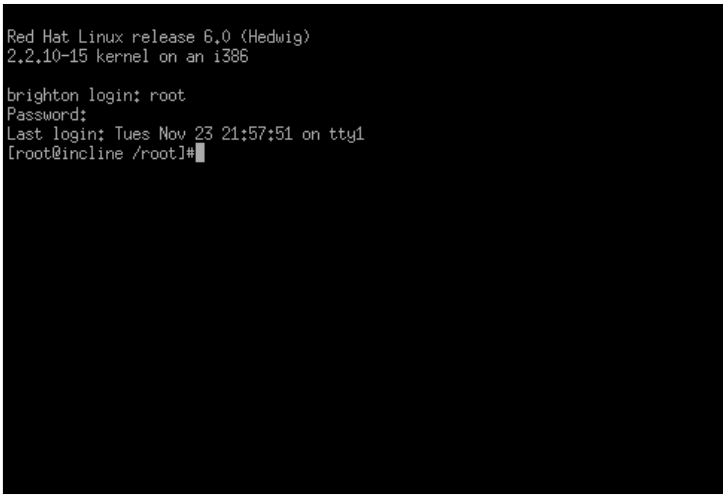

Figure 4-1 A text-mode prompt

Instead of text mode, your system may start in a graphical environment, such as Gnome or KDE. (These graphical environments are discussed in detail in Chapter 5.) Within a graphical environment, you can open a **command-line window** that permits you to enter commands at the keyboard. The command-line environment is also called a **terminal emulator window** (or terminal window) because it resembles an old-fashioned dumb terminal connection to a large computer system. Within a graphical environment, a program call **`xterm`** (pronounced

"ex-term") is sometimes used to provide a command-line window. Figure 4-2 shows a graphical environment with a command-line window open.

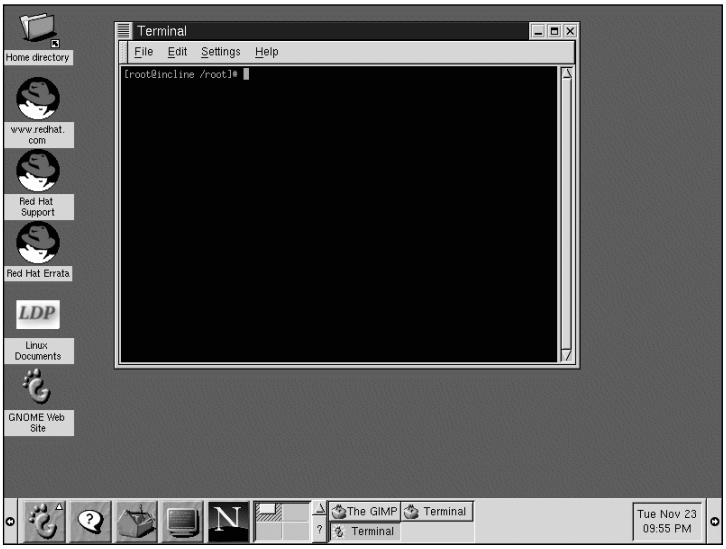

Figure 4-2 Graphical environment with a command-line window

To open a command-line window, choose the appropriate menu item within the graphical environment. For example, within the Gnome Desktop, click on the footprint icon in the lower-left corner of the window, point to `Utilities`, and then click `Gnome Terminal`. Within the KDE Desktop, click on the K icon in the lower-left corner of the window, point to `Utilities`, and then click `Terminal`.

At a command-line prompt, either in text mode or graphical mode, you can explore the Linux directory structure and learn how to use basic Linux commands.

Linux File Commands

When you first log in to Linux or open a command-line window, your working area is defined as your home directory. A **home directory** is the location where all of your personal files are stored. For the `root` user account, the home directory is `/root`. For regular users, the home directory is a subdirectory of the `/home` directory that matches your user account name. For example, if you log in as a user named `nwells`, your home directory will be `/home/nwells`. On some larger Linux systems, a different location is used for home directories, but this is rare.

In Linux, the command-line environment is called the **shell**. In Chapter 6 you will learn more details about the shell. For now you only need to know that after you type a command and press Enter, the shell will process the command. When you enter the **`pwd`** command, the shell displays your **current working directory** (that is, the directory in which you are

working). The name `pwd` stands for *print working directory*. After you first log in, the `pwd` command will display your home directory.

All of the commands presented in this section must be entered in lowercase letters. Linux is case sensitive—entering the command `PWD` produces different results from the `pwd` command. All Linux commands are lowercase.

The **`cd`** command changes the current working directory to a directory you specify. To use the `cd` command, remember the following points:

- You specify the full directory name of the directory that you want to make your current working directory. For example, `cd /home/nwells`.
- If you prefer, you can specify only the name of a subdirectory to which you want to change. For example: `cd nwells`. Note that this command does not begin with a forward slash (/).
- You can enter `cd` without any directory name after it to change to your home directory.
- To change to a directory one level higher (one level closer to the root directory, /), use the directory name **`..`** (two periods). The directory that is one level above your current directory is called its **parent directory**. For example, `cd ..` will change to the parent directory of the directory you are working in.

In DOS and Windows systems you can use the command `cd..` (without a space between `cd` and `..`) to switch to the parent directory. This command does not work in Linux. You must always include a space after the `cd` command.

Linux provides two additional commands related to directories. The **`mkdir`** command creates a new directory that is a subdirectory of the current working directory. For example, if you are in the home directory `/home/nwells`, the command `mkdir archive` creates a new subdirectory named `archive`. The full name of the new directory would then be `/home/nwells/archive`. The **`rmdir`** command removes (deletes) a directory. Note that a directory must be empty (it cannot contain any files) before you can delete it using the `rmdir` command.

The **`ls`** command lists the files in a directory. When used alone, the `ls` command only prints the names of files in a directory. But the `ls` command has dozens of options for printing additional information about files and directories. One commonly used option is `-l` (a hyphen followed by lowercase *l*). Entering the command `ls -l` prints a long list of details about each file, such as when the file was created and how many bytes it contains.

The **`touch`** command creates a new file with no data in it. If you use the `touch` command with a filename that already exists, the file is updated to show that the last time it was accessed was the moment in which the `touch` command was used. For example, if you enter the command `touch test`, a new file called `test` (containing zero bytes) is created. If you enter the

command `touch test` a second time, the internal data about the `test` file is updated to show that the file was accessed at that moment. Information about the date and time when an event occurred is stored on a Linux system in the form of a **timestamp**. Linux maintains a set of timestamps for each file and directory that define when the file was created, when it was last modified, and when it was last accessed. The `touch` command updates a file's last accessed timestamp (that is, the date and time when the file was last accessed).

Often you will need to make a copy of a file, remove it, or change its name or location. Linux provides commands for all of these operations. The **`cp`** command copies a file or directory from one location to another. You can use the `cp` command to copy a file within the same directory using a different filename, such as `cp file1 file2`. You can also copy a file to another directory using the same name. For example, the command `cp file1 /tmp` makes a copy of the file named `file1` in the `/tmp` directory (using the same filename, `file1`).

To delete a file, use the **`rm`** command (for *remove*). Be very careful using the `rm` command. When you delete a file using `rm`, the file cannot be undeleted like files placed in a trash can or recycle bin on a graphical desktop. (Linux graphical desktops provide a trash can, but the `rm` command does not use the trash can—it permanently deletes files.)

In Linux the operations of moving and renaming a file are combined in one command, the **`mv`** command. The name `mv` is short for *move*, which makes sense if you keep in mind that, in essence, renaming a file is the same as *moving* it to a different filename. For example, if you have a file named `test`, you can change its name to `retest` using this command: `mv test retest`. This places the renamed file in the same directory as the original file. To move a file, you provide a different directory name and, optionally, a different filename. The command to move the `test` file to a subdirectory named `archive` is `mv test archive`. If a subdirectory named `archive` does not exist, the `test` file will be renamed as `archive`. To move and rename a file, give a new location and a new filename. For example, to move the `test` file to the `archive` directory under the name `retest`, use the command `mv test archive/retest`.

Linux provides several utilities for viewing the contents of a file. The simplest of these is the **`cat`** command, which prints the contents of a file to the screen. For example, the command `cat test` prints the contents of the `test` file to the screen. (The `cat` command is also used to concatenate, or combine, multiple files into one larger file. This is described in Chapter 6.) The **`zcat`** command prints the contents of a compressed file to the screen. If a file ends with the letters `gz`, indicating that the file is compressed, you can use the `zcat` command to view the contents of the file without first uncompressing it. Most files do not fit on one screen, in which case the `cat` command works too quickly—you will only see the last 20 or so lines of text in a file. To view a larger file, use the **`less`** command, which prints the contents of a file one screenful at a time. You can use the arrow keys or Page Up and Page Down keys to move to different areas of the file. The Q key exits the `less` command. For example, the command `less /etc/termcap` displays a large configuration file on screen. Press Q to exit this file.

The `less` command is similar to the **`more`** command, which is used to print the contents of a file one screenful at a time. The `less` command has more features than the `more` command, but you can use either one to view the contents of files. The `less` and `more` commands are only intended to display text files (with human-readable content). You can use the `file` command to determine what a file contains. The **`file`** command prints a summary of the type of data contained in a file. For example, the command `file /etc/printcap` will tell you that the `/etc/printcap` file is a text file. The command `file /sbin/lilo` will tell you that `/sbin/lilo` is a Linux-executable program. Before you try using the `less` and `more` commands on files that do not contain human-readable text, you can check their contents with the `file` command.

This section has described only a few of the Linux commands used to work with files and directories on a command line. Table 4-2 summarizes these commands. You will have the chance to practice using these commands in the hands-on projects at the end of this chapter. You will also learn many other Linux commands in the sections and chapters to come.

Table 4-2 Commands for Managing Files and Directories

Command	Description	Example
`pwd`	Print the current working directory.	`pwd`
`cd`	Change to a different directory.	`cd /home/nwells`
`mkdir`	Make (create) a new directory.	`mkdir /home/nwells/ archive`
`rmdir`	Remove (erase) an empty directory.	`rmdir /home/nwells/archive`
`ls`	List the contents of a directory.	`ls`
`touch`	Create a new, empty file, or update the timestamp of an existing file.	`touch testfile`
`cp`	Copy a file to a new location or filename.	`cp testfile testfile.backup`
`rm`	Remove (delete) a file.	`rm testfile.old`
`mv`	Rename a file or move a file to a new location (possibly under a new name as well).	`mv testfile testfile.old`
`cat`	Display the contents of a file on screen.	`cat /etc/printcap`
`less`	Display the contents of a file on screen, one screenful at a time.	`less /etc/termcap`
`more`	Display the contents of a file on screen, one screenful at a time.	`more /etc/termcap`
`file`	Display a description of what a file contains or is used for.	`file /sbin/lilo`
`zcat`	Displays the contents of a compressed file on screen.	`zcat /tmp/report.gz`

In the examples used with these commands (and in browsing through Linux), you may already have noticed a few things about Linux filenames and directory names:

- All filenames are case sensitive. In general, most filenames contain only lowercase, but if you type uppercase letters when naming a new file or directory, they are stored as uppercase, and you must use uppercase the next time you refer to that file or directory. Thus the filenames `test`, `Test`, and `TEST` are all distinct.
- Filenames can be long, up to 256 characters, and can contain periods, numbers, and punctuation marks in addition to upper- and lowercase letters. You should not, however, try to use a forward or backward slash within a filename. For clarity, if a filename contains unusual characters such as punctuation, you might need to enclose the filename in quotation marks when you refer to the file in a Linux command.
- Filenames in Linux often don't include file extensions used in some operating systems. For example, a Linux configuration file may not have any ending, or it may end with `.conf`, or some other file extension. (A **file extension** is the last part of the filename after a period.) The names of Linux program files don't include any file extensions (such as .EXE or .COM on other systems).

In future sections and chapters you will learn how to control the commands introduced here by adding options. You will also learn about many additional useful commands.

Linux Files on a Graphical Desktop

Modern graphical interfaces provided with Linux include file manager windows that make it easy to perform most of the tasks you would otherwise perform using the text commands described in the previous section. A **file manager window** is a graphical interface that displays the contents of a directory (usually as a collection of icons) and lets you work with the files and directories using menus, mouse clicks, and dialog boxes.

The descriptions here refer to the Gnome Desktop interface, but the KDE Desktop provides almost identical functionality.

To open a file manager window in the Gnome Desktop, open the Gnome main menu by clicking on the footprint icon in the lower-left corner of the screen; then click `File Manager`. A file manager window appears like the one shown in Figure 4-3. (In KDE, choose `Home Directory` from the KDE main menu.)

The file manager window includes a list of directories on the left side. You can view the subdirectories within a directory by clicking on the small plus sign (+) to the left of a directory name. (If the directory contains no other directories, no plus sign is shown.) When you click on a directory in the list, the right side of the file manager window shows the files contained in that directory.

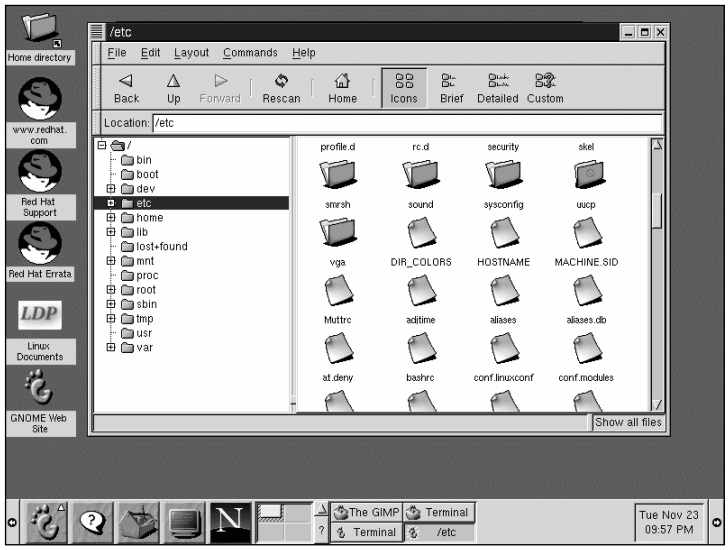

Figure 4-3 A Gnome file manager window

You can click on a file in the right side of a file manager window to perform operations on that file. For example, you can drag and drop a file's icon to another directory, either in the left side of the same file manager window or in another file manager window. You can also right-click on a file's icon to see a pop-up menu with a list of options, such as `Copy`, `Delete`, and `Move`. The `Properties` option in the pop-up menu opens a dialog box in which you can view status information about the file you selected. Figure 4-4 shows a Properties dialog box for a file within the Gnome Desktop. You will learn more about the other tabs in this dialog box later in this chapter.

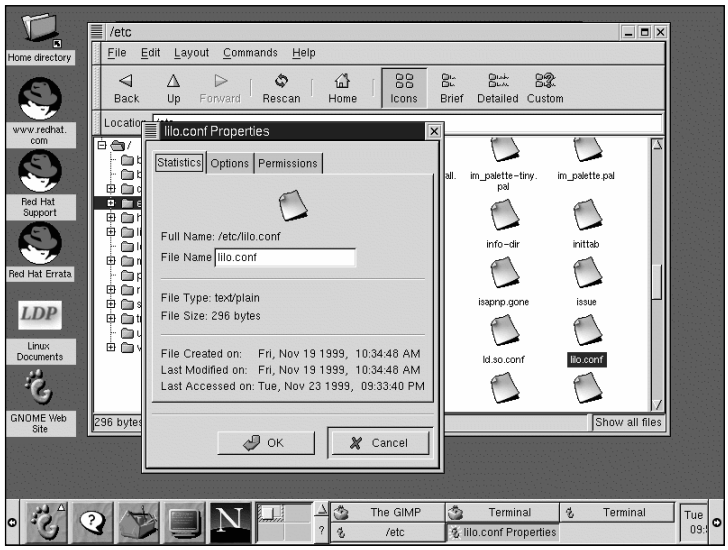

Figure 4-4 Properties dialog box viewed in the Gnome file manager

File Properties

Several properties are associated with each file in Linux. You learned earlier in this chapter about the timestamp values associated with a file, such as the date when the file was created and last accessed. (These values are visible in the Properties dialog box shown in Figure 4-4.) You can view other properties of a file or a directory using the `ls -l` command. The output of this command for two sample files is shown here:

```
-rwxr-xr-x   1 nwells   users     121024 Nov 18 14:36 newprogram
-rw-rw-rw-   1 nwells   users          0 Nov 18 15:22 test
```

Although the fields of information displayed by the `ls` command are not labeled, you should be able to recognize the username `nwells` in the sample output. Each file has a username associated with it. This user is the owner of the file. Whenever you create a new file or directory, your username is assigned as the owner of that file. Each file and each directory is also assigned a group; the group is then said to *own* that file. A **group** is a named account that consists of a collection of users. Each member of a group has access to files owned by that group. (You will learn about users and groups in detail in Chapter 8.) In the sample listing above from the `ls` command, the `users` group is assigned to the files. Depending on how your Linux distribution is configured, all of the regular user accounts on a Linux system might be members of the `users` group.

The `root` user can execute the **`chown`** command to change the ownership of a file or directory. To use this command, type `chown`, followed by the username and group (separated by a period) that you want to assign to the file or directory. Next, type the name of the file or directory. For example, the following command changes the owner of the file `test` to `jtaylor` and the group assigned to the file `test` to `managers`:

```
chown jtaylor.managers test
```

The user and group that you assign to a file must already exist on the Linux system, as described in Chapter 8.

The owner and group assigned to a file determine who can access the file. The term **file permissions** refers to the type of access that a user has to a file or directory on the Linux system. If you have used other operating systems, you may have heard terms such as *file rights* or *access permissions*. These terms are synonymous with file permissions. Linux file permissions provide adequate security to protect access to files and directories, but they are not as detailed as those provided by other operating systems. In Linux, only three different file permissions can be assigned: read, write, and execute.

- **Read permission** allows a user to read the contents of a file or browse the files in a directory. This permission is designated in file listings by a letter `r`.
- **Write permission** allows a user to add or change information in a file or create files within a directory. This permission is designated in file listings by a letter `w`.
- **Execute permission** allows a user to launch a file as a program or see files within a directory. This permission is designated in file listings by a letter `x`.

Each of these three possible permissions can be assigned in three different ways:

- **User permissions** always apply to the owner of a file or directory.
- **Group permissions** always apply to members of the group assigned to a file or directory.
- **Other permissions** always apply to all users on the Linux system who are not the owner of the file or directory in question and are not members of the group assigned to the file or directory.

The three permissions assigned in three ways create a total of nine permissions that can be assigned to any file or directory in Linux. These nine permissions are shown on the left side of the output of the `ls -l` command. Consider this sample output from the `ls -l` command:

```
-rwxr-xr-x   1 nwells   users   121024 Nov 18 14:36 newprogram
```

The far left character of the output (a hyphen in this example) indicates the type of item you are viewing. A hyphen indicates a regular file. The letter `d` indicates a directory. The next nine characters represent three permissions for the file's owner (user permissions), three permissions for members of the file's assigned group, and three permissions for all other users on the system. As a further example, consider a few common arrangements of Linux file permissions. A utility such as `fdisk` or `less` has the `root` user assigned as the file's owner and has file permissions that appear like this in a file listing using `ls -l`:

```
rwxr-xr-x
```

These indicate that the file's owner (`root`) can write to (alter) the file, and that everyone else on the system (both group members and other users) can read the file and execute it as a command. A common set of permissions assigned to files that you create is shown here:

```
rw-r--r--
```

You, as the owner of the file, can read and write to it. Others on the system can read the file's contents but cannot alter the file.

You can use the **`chmod`** command to change the permissions assigned to any file or directory that you own. If you are logged in as `root`, you can use `chmod` to change the permissions of any file or directory on the system. To use the `chmod` command, include the type of permissions you want to change (user, group, or other, entered as `u`, `g`, or `o`), followed by a plus or minus sign to add or remove permissions, followed by the permissions you want to add or remove (`r`, `w`, or `x` for read, write, or execute). For example, to add the write permission for other users to the file `test`, use this command:

```
chmod o+w test
```

You can also set specific permissions using an equal sign. For example, to set the permissions for members of the group assigned to the file `test` to read and write, use this command:

```
chmod g=rw test
```

System administrators often use a different syntax with the `chmod` command. This alternative syntax is easier to use once you are familiar with it, but it's more challenging to learn. In this alternative syntax, each of the sets of three permissions (for user, group, and other) is represented by a number from 0 to 7. The three possible permissions (read, write, and execute) are assigned values of 4, 2, and 1, respectively. Now suppose that a system administrator wanted to grant read (4) and write (2) permission for the user, read (4) permission to the group, and no permissions to other users. The first digit used in `chmod` is 6, the sum of 4 and 2; the second digit is 4; the third digit is 0 (no permissions are granted). So the command would look like this:

```
chmod 640 test
```

Using the same method, if the system administrator wanted to assign read (4), write (2), and execute (1) permissions to the user, read (4) and execute (1) permissions to the group, and the same to other users, the command would look like this:

```
chmod 755 test
```

Although this method may appear strange at first, you should become familiar with it because you will see it used often by experienced system administrators on all UNIX systems. You will also discover that only a few combinations of file permissions are commonly used. Once you are familiar with the three-digit code for those commonly used sets of permissions, using the three digits is easier than entering all the letters with a plus, minus, or equal sign.

Conversely, using a graphical environment like Gnome or KDE provides a much easier method of setting file permissions. Within the Properties dialog box described previously, the Permissions tab includes check boxes in which you can activate or remove any of the nine permissions described in this section. Figure 4-5 shows the Permissions tab in the Properties dialog box of the Gnome file manager.

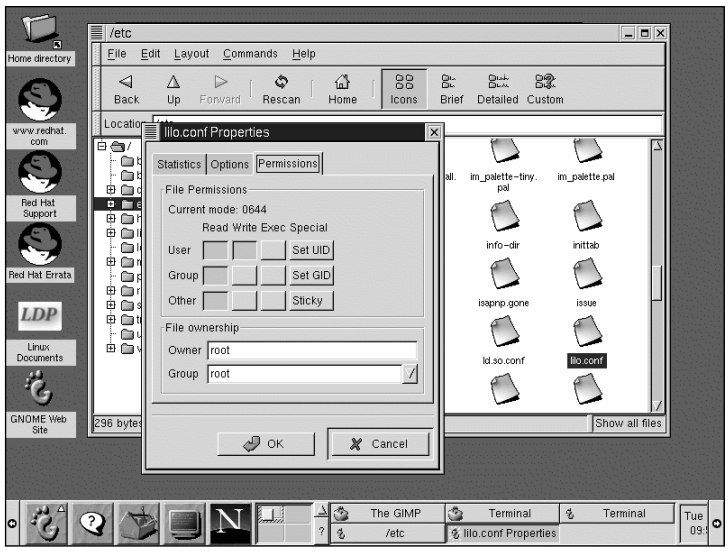

Figure 4-5 Permissions tab of the Properties dialog box

The Permissions tab includes additional security information that is beyond the scope of this book.

Each time you create a new file, a default set of file permissions are assigned to the file. On most Linux systems, the permissions for a newly created file look like this:

```
rw-r--r--
```

The permissions above assign you as the file's owner, and give you permission to read and write to the new file. Others can only read the file. The **`umask`** command determines the file permissions assigned when you create a new file. The `umask` command is executed automatically when you log in to Linux. You can alter the default permissions assigned to a new file by executing the `umask` command again at any time. The `umask` command uses three-digit codes similar to the `chmod` command. Assume that the default permissions without using `umask` would look like this:

```
rw-rw-rw-
```

The value provided with the `umask` command disables one or more of these standard permissions, using the same numbers assigned to each permission as the `chmod` command (4 for read, 2 for write, and 1 for execute). Thus, a standard `umask` command like this will disable the write permission for the group and other categories:

```
umask 022
```

This results in the default file permissions shown here.

```
rw-r--r--
```

MANAGING SOFTWARE PACKAGES

To run any program in Linux, you simply enter the program's name at a command-line prompt. The shell then loads and executes the command. This is true for command-line utilities, such as `cp` and `ls`, and also for programs, such as WordPerfect for Linux (whose program file is named `xwp`) or Netscape Communicator (whose program file is named `netscape`). Some programs allow you to add parameters or options along with the command to run the program. A **command-line parameter** is an additional piece of information (besides the name) that is included on the command-line when a program is started. The shell passes the command-line parameter to the program to control how the program operates. In the previous section you learned about one example of a command-line parameter: the `ls -l` command. The `-l` command-line parameter is passed to the `ls` program to instruct it to print a long-format list of files. Command-line parameters are also called **command-line options**.

To execute a program, Linux must be able to locate it within the Linux directory structure. For example, when you enter the command `cp`, Linux must be able to locate the `cp` program file in the directory where it is stored: `/bin`. To do this, Linux uses an environment variable called `PATH`. A **variable** is a memory location used by a program to store a value, such as a

number or a word. Each variable is assigned a name so that the program can access the value by referring to the name. **Environment variables** are variables that are defined by the Linux shell so that all programs can access their values. The `PATH` environment variable includes a list of all the directories where programs on the system are normally located, such as `/bin`, `/usr/local/bin`, and others. When you enter a program name to execute, the shell searches each of the directories named in the `PATH` environment variable until the program name is found. Then the program is loaded and executed. If the program cannot be found in any of the directories listed in `PATH`, the shell returns an error. For example, attempting to execute a nonexistent program called `makedir` (the real command is `mkdir`) causes the shell to print the following error message:

```
bash: makedir: command not found
```

You can view the value of the `PATH` environment variable (and thus see where all the programs on your system are stored) by using the `echo` command. The **`echo`** command prints to the screen (that is, displays on the screen) text you specify. For example, the command `echo This is a test` displays this on the screen:

```
This is a test
```

Whenever the shell detects the name of an environment variable preceded by a dollar sign, such as `$PATH` , the shell replaces the name of the environment variable with the numbers or letters stored in memory under that variable's name. Using the `echo` command you can display on the screen the numbers or letters stored in an environment variable. Use the command `echo $PATH` to display on the screen the value of the `PATH` environment variable. Sample output of the command `echo $PATH` is shown here:

```
/usr/bin:/bin:/usr/local/bin:/usr/bin/X11:/usr/X11R6/bin:/home/nwells/bin
```

The value of the `PATH` environment variable differs slightly based on which version of Linux you are using, though the value will always be similar to the preceding output. Notice in the output above that each of the directories where programs are stored is separated by a colon (:).

To run a program stored in a directory that is not named in the `PATH` environment variable, use the full pathname of the program. For example, if you have a new program called `kpacman` stored in your home directory (which is not part of the `PATH` variable), use a command like this to start the program:

```
/home/nwells/kpacman
```

You can also use a single period to refer to the current directory. (Remember, use two periods to refer to the parent directory.) So to run a program called `kpacman` that is stored in your current directory (whatever directory you happen to be working in), the command would look like this:

```
./kpacman
```

In Chapter 6 you learn more about creating and using `PATH` and other environment variables.

Function Libraries

Many Linux programs require the same underlying functionality to complete tasks. For example, most programs need to open files on the hard disk. Many programs also need to read information from the keyboard or write results to the screen. Each of these tasks is called a function in computer programming jargon. A function is a small task that a computer program performs. The average programmer could probably list hundreds of functions commonly required by his or her programs.

Rather than having to include all of these functions in every program, programmers can assume that the necessary functions are already included in a Linux system, in the form of a function library. A **function library** is a file that contains a collection of commonly used functions for any program to use as it runs. Dozens of these libraries are installed when you install Linux. Function libraries help a Linux system conserve system resources (such as memory), because one function library can be used by multiple programs at the same time.

The directories `/lib` and `/usr/lib` contain most of the libraries used by Linux programs. A few of the commonly used libraries are listed in Table 4-3. As you explore the directories `/lib` and `/usr/lib`, notice that the library files all begin with the name `lib` and end with the file extension `.so`, followed by a version number. (The version number is not included in Table 4-3.) The file extension `.so` stands for *shared object*, because the libraries can be shared among many programs. When you install new libraries on your system, they are normally placed in the `/usr/lib` directory. Some parts of the Linux system, however, such as the KDE Desktop or the X Window System, have separate directories for dedicated library files. For example, `/opt/kde/lib` for KDE and `/usr/X11R6/lib` for the X Window System are two standard locations.

Table 4-3 A Few Common Function Libraries

Library	Purpose
`libc`	Standard programming functions in the C programming language
`libm`	Mathematical functions
`libext2fs`	Functions to access an `ext2`-format (native Linux) hard disk
`libtermcap`	Functions for working with terminals (controlling the character format and output on character-mode screens)
`libcrypt`	Functions to encrypt passwords for managing user logins
`libgif`	Functions used when processing gif-format graphics

As you work with new programs in Linux, sometimes the libraries needed to run a program are not included on your Linux system. To fix this you must determine which libraries are needed and add them to your system. The **ldd** command lists the function libraries that a program uses. For example, to view the libraries used by the `ls` command, use this command:

```
ldd /bin/ls
```

This command results in only two lines displayed on the screen, indicating that the `ls` command uses only two libraries. Larger, more complex programs (such as graphical programs) will use many more libraries. For instance, `gtop` (a Gnome system administration utility described in Chapter 10) uses a number of libraries. Consider the output of the command `ldd /usr/bin/gtop`.

```
libgnomeui.so.32 => /usr/lib/libgnomeui.so.32 (0x40019000)
libart_lgpl.so.2 => /usr/lib/libart_lgpl.so.2 (0x400d1000)
libgdk_imlib.so.1 => /usr/lib/libgdk_imlib.so.1 (0x400de000)
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x40102000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x4010b000)
libgtk-1.2.so.0 => /usr/lib/libgtk-1.2.so.0 (0x40122000)
libgdk-1.2.so.0 => /usr/lib/libgdk-1.2.so.0 (0x40235000)
libgmodule-1.2.so.0 => /usr/lib/libgmodule-1.2.so.0 (0x40266000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40269000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40276000)
libgnome.so.32 => /usr/lib/libgnome.so.32 (0x4031a000)
libgnomesupport.so.0 => /usr/lib/libgnomesupport.so.0
libesd.so.0 => /usr/lib/libesd.so.0 (0x40332000)
libaudiofile.so.0 => /usr/lib/libaudiofile.so.0 (0x40338000)
libm.so.6 => /lib/libm.so.6 (0x40345000)
libdb.so.2 => /lib/libdb.so.2 (0x40362000)
libglib-1.2.so.0 => /usr/lib/libglib-1.2.so.0 (0x40370000)
libdl.so.2 => /lib/libdl.so.2 (0x40390000)
libgtop.so.1 => /usr/lib/libgtop.so.1 (0x40393000)
libgtop_sysdeps.so.1 => /usr/lib/libgtop_sysdeps.so.1
libgtop_common.so.1 => /usr/lib/libgtop_common.so.1 (0x403a4000)
libgdbm.so.2 => /usr/lib/libgdbm.so.2 (0x403a9000)
libc.so.6 => /lib/libc.so.6 (0x403af000)
libz.so.1 => /usr/lib/libz.so.1 (0x4049d000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

In the next section you learn how to add missing libraries to a Linux system using the `rpm` command.

Adding and Removing Software Packages

In Chapter 3 you learned that a single software package file can contain all of the files and configuration information needed to set up a new application or collection of utilities. You also learned that the Red Hat Package Manager (rpm) data format is the most popular type of software package for Linux. Files in rpm format (each with a file extension of `.rpm`) are copied

to the system during the Linux installation process. After the installation is complete, you can use the **rpm** command to manage all of the rpm software packages on a Linux system.

The many options of the `rpm` command allow you to maintain a database of all the software installed on the Linux system. You can query this database to learn about what software is installed, what version of a software package you are using, and other information. You can also use the `rpm` command to install new software packages or remove software packages from the system.

To see if a package is installed on the system, use the `-q` option. To use this option you must know the name of the package. For example, to see if the Apache Web server is installed, use the command `rpm -q apache`. The response tells you either the complete name and version number of the installed package or that the package is not installed. The command `rpm -qa` lists all the packages installed on the system. You can use this command with the `less` command to browse through the list. The command would look like this:

```
rpm -qa | less
```

You can install new packages in rpm format using the `rpm` command. Linux CDs that are based on rpm-format packages will contain hundreds of rpm files that you can use to install additional software on your system. For example, the installation option that you selected when installing Linux might not have included the graphical program called Gimp. Once you have mounted your Linux CD-ROM drive with a command such as `mount /mnt/cdrom` (Chapter 9 provides details on this step), you can change to the directory containing the rpm files and use the `rpm` command to install `Gimp`. On Red Hat Linux, the necessary commands are as follows. (The directory in the second command and the version number in the final command will be different on other Linux systems.)

```
mount /mnt/cdrom
cd /mnt/cdrom/RedHat/RPMS
rpm -Uvh gimp-1.0.4-3.i386.rpm
```

After the last command shown here is executed, a list of hash marks (`#####`) appears on the screen, indicating that the files in the software packages are being installed. The `-U` option on the `rpm` command line indicates that you want to upgrade the package. Thus, if a package of the same name is already installed on the system, the `-U` option updates the package using the newer package's files.

You can also delete a package from the Linux system using the `-e` option (for *erase*). Note that all information about a package is completely erased when you use the `rpm -e` command. This means that once you delete a package, you can only reinstall it using the original rpm file from the Linux CD (or from the Internet). For example, the following command erases the Gimp program from the Linux system:

```
rpm -e gimp
```

When using `rpm` to install a package, you must specify the complete filename (ending with `.rpm`). Otherwise `rpm` will not be able to locate the software package in the directory structure. When using the `-q` or `-e` option, you only need to specify the package name, without a version number or `.rpm` extension, to locate the package within the internal rpm database.

The `rpm` command includes dozens of more complex options, which you can review by entering `rpm` without any command-line parameters after the command.

Several graphical programs are available to help you manage rpm software packages. These programs use the `rpm` command in the background as you select menu items and work in dialog boxes, which provide an easy-to-use interface to the many functions of the command. For example, you can list a description of a package by using the command `rpm -qi`, or list all the files in a package by using the command `rpm -ql`, or show which package contains a certain file by using the command `rpm -qf` *`filename`*. But rather than memorize all of these commands immediately, you can use the menu items and dialog boxes provided by one of the rpm graphical tools.

One such graphical tool is the `GnoRPM` package management utility. To open it from the Gnome Desktop, click the footprint icon, point to `System`, and then click `GnoRPM` (see Figure 4-6). The left side of the window shows a list of categories into which packages have been divided for easy reference. The right side of the window displays an icon for each software package included in the category that you select. You can right-click on a package icon to display a list of options related to that package.

The KDE Desktop also includes a package management tool called `kpackage`. You can start this program from the `System` menu on most KDE Desktops. (Some systems use a different location for `kpackage`; for instance, Caldera OpenLinux places `kpackage` on the `COAS` submenu.) Within `kpackage`, categories are displayed on the left. Click on a category to open a list (also on the left side of the window) of the packages in that category. Click a package name to display information about that package in the right side of the window. A description of the package and a list of every file contained in the package are available on two tabs on the right side of the window. Figure 4-7 shows the `kpackage` utility displaying information for one software package.

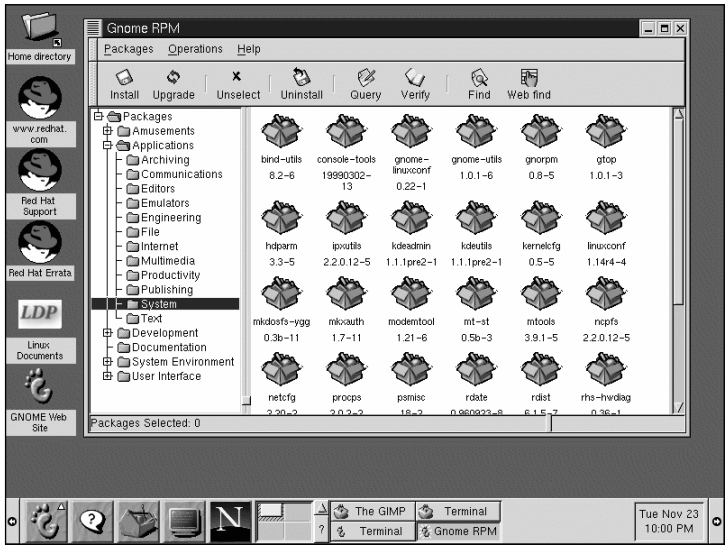

Figure 4-6 The `GnoRPM` package management utility

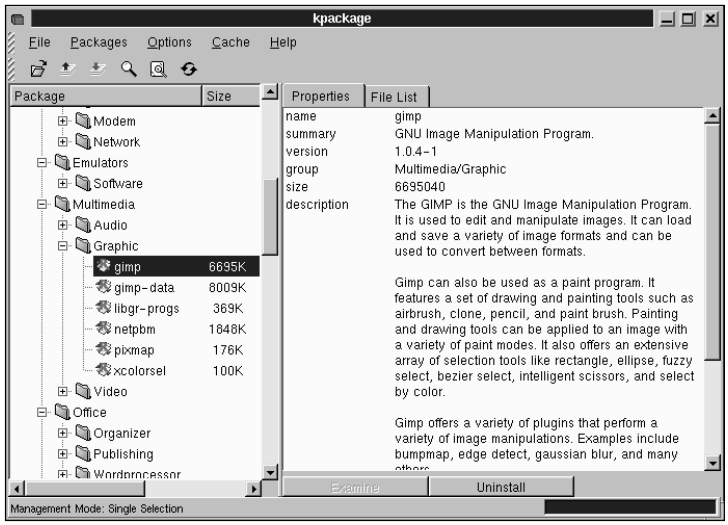

Figure 4-7 The `kpackage` utility for software package management

Both `GnoRPM` and `kpackage` provide menu items that let you search for a package with a certain name, search for a package containing a certain file on the system, or install and uninstall software packages. They do not provide access to all the features of the `rpm` command, but they handle most tasks that you are likely to need as a system administrator. The `kpackage` utility

also lets you manage different types of software packages, such as those in `.deb` format used by the Debian Linux distribution.

Both `GnoRPM` and `kpackage` can be started from the command line as well. Within a terminal window, enter the command `gnorpm` (or `kpackage`).

Using tar Archive Files

Although many Linux systems use rpm software packages, another common format, a tar archive, is used for much of the Linux software that you might see on the Internet. The **`tar`** command is used to create a single file that contains many other files, often compressed to save space. A **tar archive** is a file created by the `tar` command and has a file extension of `.tar`. When a tar archive has been compressed using the standard Linux compression utility, `gzip` (pronounced gee-zip), the file is sometimes called a *gzipped tarball*. These compressed tar archives have the file extension `.tgz` or sometimes `.tar.gz`.

A tar archive does not provide the functionality of an rpm software package. The tar archive is simply a collection of files within one directory tree. No database of installed tar archives is maintained, and no special tools support the features of rpms when using tar archives. But tar archives are supported on virtually every UNIX and Linux system in the world, so they make a convenient method of sharing files across the Internet. As you explore Internet sites containing Linux programs, such as *ftp://metalab.unc.edu/pub/Linux*, you will see many tar archives. After you have downloaded a tar archive, you can extract its contents using the `tar` command. For example, to extract the contents of an archive file named `program.tgz`, use this command:

```
tar xvzf program.tgz
```

The `tar` command is used extensively in Linux for creating backups of files on the system. The name `tar` comes from *tape archive*, because data backups were formerly always written to a tape back-up device. (Now devices such as writeable CD drives are also used.) Chapter 14 describes how to use the `tar` command to create and manage backups on Linux.

A tar archive is not the only type of file that can be compressed on Linux. You can also use the **`gzip`** command to compress any file on the system. For example, to compress the file `large.doc`, use this command:

```
gzip large.doc
```

The preceding command transforms the file `large.doc` into a compressed file called `large.doc.gz`. The **`gunzip`** command uncompresses a file that you have compressed using `gzip`. For example, to uncompress the `large.doc.gz` file, use the command `gunzip large.doc.gz`. The resulting file is named `large.doc`.

When you use the `tar` command to extract the contents of a tar archive, the tar archive remains intact. When you use the `gzip` or `gunzip` command to compress or uncompress a file, the original file is altered (compressed or uncompressed), and its name is changed accordingly.

The amount that the `gzip` program compresses a file depends on the type of data contained in the file. Text files and some types of graphics files may be compressed by 70 to 90%. That is, a file of 1 MB may be compressed to only 100 KB to 300 KB. Other types of data may only be compressed by 10 to 20%.

The `gzip` utility is the most commonly used compression tool on Linux systems, but it is not the only one available. All of the compression utilities are listed in Table 4-4.

Table 4-4 Compression Utilities in Linux

Compress/uncompress utility	Description
`gzip` and `gunzip`	Provides good compression ratios. Use this tool for most cases; it is the most commonly used compression tool on Linux. Compressed files have the extension `.gz`.
`zip` and `unzip`	Compresses multiple files into one file, much like a compressed tar archive. Compatible with `pkzip` and `WinZip` on other operating systems. Use this program to share files with users on non-Linux systems. Compressed files have the file extension `.zip`.
`compress` and `uncompress`	Provides poor compression compared to `gzip` (files are not as small when compressed with the `compress` command). It is an older utility supported on almost all UNIX systems. Compressed files have the extension `.z`.
`bzip` and `bunzip`	Provides excellent compression (creates very small files), but it is a newer utility that is not widely used yet. Limit your use to sharing files with those you know have `bzip`. Compressed files have the extension `.bz2`.

THE LINUX KERNEL

In Chapter 1 you learned the definition and function of an operating system kernel. In this section you learn more about how to manage the Linux kernel. The Linux kernel continues to be managed by Linus Torvalds, the original creator of Linux. New features are regularly added to the Linux kernel, which is then integrated into new versions of Linux products such as SuSE Linux, Turbo Linux, and Red Hat Linux. As new features are added to the kernel, Linus Torvalds (and the developers with whom he works) assign new version numbers to the kernel.

You can learn more about the current status of Linux kernel development by visiting the Web site *www.linuxhq.com*. The latest Linux kernel is always available for download from this site, or from the FTP site *ftp.funet.fi* in Finland.

Learning About Your Kernel

After installing Linux you can use the `uname` command to learn about the kernel that is running your particular system. The `uname` command provides the kernel version, as well as additional information about the operating system. Two useful options of the `uname` command are `-r` and `-v`. The `-r` option (for *release*) displays the Linux kernel version number, followed by a release number. The **release number** is a number assigned by the company that prepared the Linux product. The release number allows the company to track how many times the kernel has been altered before shipping the final product. For example, using the `-r` option on your system might result in something like:

```
2.2.10-15
```

Here the kernel version is `2.2.10`, and the release number is `15`.

Each kernel also has a timestamp associated with it, which indicates the date and time when the kernel was created. As you begin to create your own Linux kernels (later in this chapter), you will need to use the `uname -v` command to check the timestamp of the kernel that is currently controlling the system. For example, after installing a new Linux system, the `uname -v` command might show something like this:

```
#1 Fri Nov 19 10:25:20 PST 1999
```

After you have created a new kernel with different features enabled, and rebooted the system to start the new kernel, the `uname -v` command would show a different date and time, letting you know that the new kernel was active.

You can also use the command `cat /proc/version` to print the kernel version to the screen. Sample output from this command is shown here. It includes both the version number and the timestamp.

```
Linux version 2.2.5-15 (root@incline.xmission.com) (gcc ver-
sion egcs-2.91.66 19990314/Linux (egcs-
1.1.2 release)) #1 Fri Nov 19 10:25:20 PST 1999
```

Kernel Modules

One of the most useful features of Linux is its ability to add and remove features of the kernel without restarting the computer. Linux **kernel modules** are files containing computer code that can be loaded into the kernel or removed from the kernel as needed. Some features of the kernel are built in, but many others are available by adding modules to a Linux kernel. For example, you can add kernel modules to provide any of the following:

- Support for a network adapter card
- Support for a SCSI hard disk controller card
- Networking features such as firewall capability
- The ability to access other types of file systems, such as data stored in Windows NT
- Support for a sound card

Some kernel modules may be automatically loaded into the kernel based on the configuration you set up during the Linux installation. The **lsmod** command lists the modules that are installed in the Linux kernel. The names of modules are often very cryptic, but some are recognizable. For example, the sound module is used for sound card support, and the scsi module is part of the support for SCSI hard disk controllers. Sample output of the lsmod command is shown here:

```
Module                  Size  Used by
nfsd                  150936   8  (autoclean)
lockd                  30856   1  (autoclean) [nfsd]
sunrpc                 52356   1  (autoclean) [nfsd lockd]
pcnet_cs                7456   1
8390                    5920   0  [pcnet_cs]
ds                      5740   2  [pcnet_cs]
i82365                 21956   2
pcmcia_core            39720   0  [pcnet_cs ds i82365]
```

Each kernel module is stored as a file on the hard disk. When the module is added to the kernel, it is copied from the hard disk to memory as part of the kernel. Figure 4-8 shows how kernel modules relate to the Linux kernel.

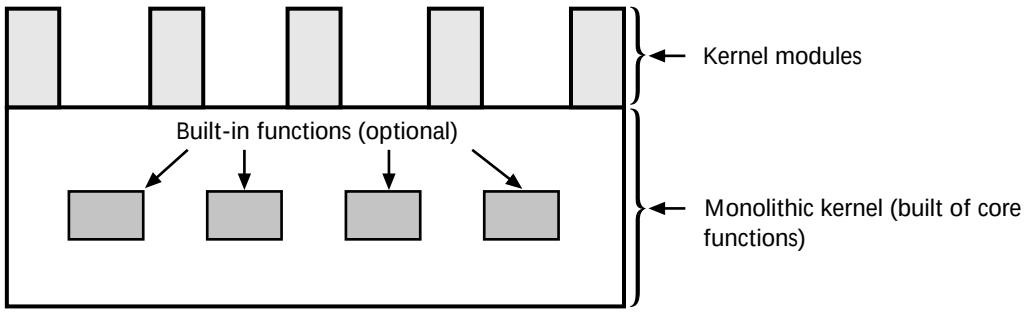

Figure 4-8 Linux kernel modules and the kernel itself

Adding and Removing Modules

If you need to add a feature to the Linux kernel (such as support for a sound card or a networking feature), you can use the insmod command. The **insmod** command copies a module file from the hard disk and adds it to the Linux kernel running in memory. For example, the command insmod sound adds the sound module to the kernel.

Sometimes one module requires another module in order to function correctly. For example, to use the sb module to support a SoundBlaster card, you must first load the sound module. The **modprobe** command loads a module with all of its required supporting modules. For example, you use the command modprobe sb to load the SoundBlaster module and all the modules that it requires to function correctly. Using modprobe is the preferred method of adding kernel modules.

The **rmmod** command removes a module from the kernel. The module remains available on the hard disk so that you can insert it again later using insmod or modprobe.

Some modules require specific hardware information in order to function correctly. For example, when you add a module to support a network adapter card, you may need to include information about the card's IRQ. (This information might have already been supplied during the Linux installation.) **Module parameters** provide information needed by a module to locate system resources. When using the `insmod` or `modprobe` command, you should add module parameters after the module name. For example, to add a kernel module for an NE2000 network adapter (which requires a module called `ne2`), the following command would specify the hardware's IRQ and I/O port address:

```
modprobe ne2 irq=11 io_port=0x330
```

The `0x` at the beginning of the last number in the preceding example indicates a hexadecimal value.

When you execute the `insmod` or `modprobe` command, the module attempts to communicate with any related system hardware as it is being loaded. If the module loads successfully, you see no feedback on the screen. If a problem occurs, you see a message stating that the module could not be loaded or could not be initialized. Such a message means that the module parameters were either incorrect or inadequate.

Locating Modules

When you see a module name (something like `sb` or `aic7xxx`), it's difficult to know which devices or kernel features that module provides support for. In addition, the module parameters that are supported by each module are difficult to find. You can always experiment with different modules until you finally locate the correct module for hardware that is not working correctly. However, it's more efficient to contact the vendor of your Linux system and ask the technical support representative which module and parameters to use. (You will still need to determine the values for the required parameters based on your computer's hardware configuration.)

The source code for the modules contains detailed information about what devices are supported by each module and what parameters are supported by the module, but reading the source code of a module (not to mention simply locating the correct module in the first place) can be tedious and time consuming. However, if you do want to review a module's source code, simply load any of the files from the following directory into a text editor: `/usr/src/linux/modules`.

The modules that are available on your system are stored in the directory `/lib/modules/`*`version`*, where *`version`* is the version number of the Linux kernel on the system. For example, `/lib/modules/2.2.12` is one possible directory name. Within this directory are several subdirectories, each of which contains dozens of modules. Example subdirectories include `ipv4` for networking features, `net` for network adapter card support, and `video` for Linux video support. The `insmod` and `modprobe` commands automatically search these directories for the correct module file.

Automatically Loading Modules

You can use the `insmod` or `modprobe` command to load kernel modules into memory as part of the Linux kernel after the Linux system has started. But it is more convenient to have the necessary modules loaded automatically as the system starts. You can do this using several configuration methods. Before you automate module loading, you need to verify that you can add the module successfully using `insmod` or `modprobe`.

The first method for automating the process of loading a module is to add the appropriate `insmod` or `modprobe` command (with any needed module parameters) to the end of the `/etc/rc.d/rc.local` file. This file is executed each time the system is started (as described later in this chapter). When you add a command, such as the following `modprobe` example, to the end of the `rc.local` file (using any text editor program), the command is executed automatically as the system boots.

```
modprobe ne2 irq=11 io_port=0x330
```

The second method for installing modules is to let the `kerneld` program attempt to load modules automatically when they are needed. Red Hat Linux is designed so that system administrators can easily use this method to automate module loading. The `kerneld` program watches for programming requests that cannot be handled without a new module. It then loads the required module automatically. Keep in mind, however, that the `kerneld` program may not be able to determine the module you need. In this case, you can add a module name to the file `/etc/conf.modules` in Red Hat Linux. This causes the module to be automatically loaded by the Linux start-up process.

Yet a third method relies on a list of modules that are needed for the system to operate. All the modules in the list are loaded at start-up time. Caldera OpenLinux is designed to have system administrators use this method. (OpenLinux includes scripts that a system administrator can modify to easily implement this method of module loading.) The file containing the list of modules is stored in the directory `/etc/modules/`*`version`*, where *`version`* is the kernel version number, such as `2.2.10`. The file containing the list of modules is named for the timestamp of the kernel. For example, suppose the output of the `uname -v` command gives the following output:

```
#1 Fri Nov 19 10:25:20 PST 1999
```

In this case, the list of modules to load for that kernel is located in the following file (using the example kernel version of 2.2.10):

```
/etc/modules/2.2.10/#1 Fri Nov 19 10:25:20 PST 1999
```

Because this filename contains spaces, you need to enclose it in quotation marks when using it in a command. For example, to view the list of modules to be autoloaded in OpenLinux, use this command:

```
cat "/etc/modules/2.2.10/#1 Fri Nov 19 10:25:20 PST 1999"
```

Modifying the Linux Kernel

Although the Linux kernel provides many features and support for many types of hardware, you may discover situations that make it necessary to recompile the source code of the Linux kernel. Examples of when this might be necessary include the following situations:

- You need to add a feature to your system that Linux supports but which is not activated in the kernel that you installed by default. For example, you may need to add support for multiprocessing systems (such as dual-Pentium computers).
- You need to add built-in support for a hardware device that is currently supported only by adding a module to the kernel. For example, some types of SCSI hard disks require built-in kernel support in order to boot the Linux system from those hard disks.
- You want to use an updated version of the Linux kernel to add new features or fix a problem that was discovered with your current version.

In any of these cases, you can recompile the Linux kernel from its source code into a new kernel. The file `vmlinuz` contains the Linux kernel. It is usually located in the `/` directory or in the `/boot` directory. Before you can work with the kernel source code, you must verify that it is installed on your system. If you are using Red Hat Linux, you should find the source code in a single rpm package named `kernel-source`. You can use the `rpm` command with the `-q` option to see if this package is installed, or use the `rpm` command with the `-Uvh` options to install the package from CD if needed. Other versions of Linux may use different names for the source code package. For example, the source code on Caldera OpenLinux is stored in two files named `linux-source-common` and `linux-source-i386`.

Once the source code is installed, you can explore the source code files in the directory `/usr/src/linux`. When you install the kernel source code from the CD that you used to install Linux, and then recompile the source code to create a new kernel file, you are not changing the version of the kernel. You are only changing the features that are activated in the kernel. For example, if the kernel installed with a Linux system is version 2.2.10, the version after recompiling the kernel remains 2.2.10, but the timestamp of the new kernel will be different because it was compiled at a different time.

Before you can recompile the kernel, you must configure the features you want to include in the kernel by using a menu-based utility provided with all Linux systems. To start this utility, first verify that the kernel source code is installed on your system. Then enter these two commands:

```
cd /usr/src/linux
make menuconfig
```

Figure 4-9 shows the menu-based kernel configuration tool. Use the arrow keys and other keys as directed in this interface to select features of the kernel that you want to activate. Each item in the initial screen leads to a submenu of options. Over 1000 configuration options are available. A default setting based on your current kernel is used for any settings that you do not alter. When you have made all the changes that you want to make in the

kernel configuration, choose the `Save and Exit` option on the main menu. A hidden configuration file is then created based on your selections.

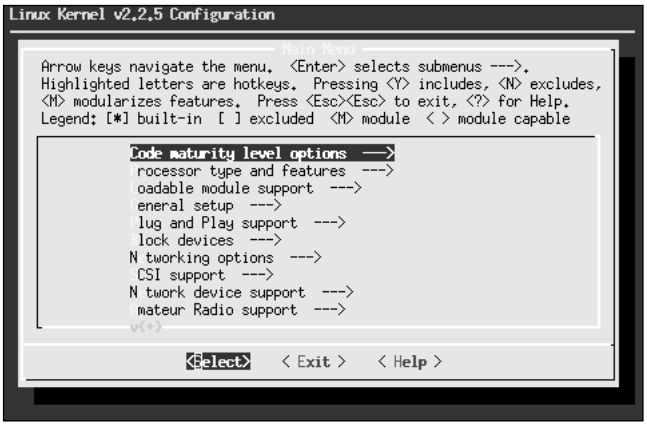

Figure 4-9 The menu-based kernel configuration screen

A graphical configuration tool is also provided with the Linux kernel. You can start that tool by entering the command `make xconfig`. You can execute the `make menuconfig` command from any command line.

After completing the configuration, you must compile the kernel source code based on the new configuration. To do this, verify that the `/usr/src/linux` subdirectory is the current directory, and then enter the following command:

```
make dep; make zImage
```

This command begins the compilation process, which can take from 5 minutes to several hours, depending on the speed of your system. A typical Pentium-class system should take 15 minutes or less. As parts of the kernel are compiled, you see hundreds of lines of text scroll on the screen.

Other compilation options are available. The commands shown in this section are the simplest means of creating a new kernel file. The file is called `zImage` and is located in the subdirectory `/usr/src/linux/arch/i386/boot`. You can copy this file to the `/` or `/boot` directory of the system and set up the LILO boot manager to use the new kernel when you restart your system. You must set up LILO to use the new kernel because the LILO program determines which kernel file is loaded when you boot the system. Configuring the LILO boot manager is discussed at the end of this chapter.

Sometimes a Linux vendor such as Red Hat Software will prepare updated kernels as rpm packages. In this case you can use the `rpm` command to update the kernel without following the steps in this section. Check the instructions from your Linux vendor carefully before attempting to install a new Linux kernel using the `rpm` command.

THE INITIALIZATION PROCESS

When you turn on a computer, many things must occur before the operating system is loaded and ready to accept commands and execute programs. The following sections describe the main steps that Linux takes to initialize the system each time you turn it on. Some of the details of this process vary among Linux distributions, but most Linux systems use a process very similar to the one described here for Red Hat Linux.

Booting the Kernel

When you first turn on the computer, the LILO boot manager displays a prompt that reads `boot:`. At this prompt you can choose which operating system to load. (If the computer has only Linux installed, you can only choose Linux.) Normally, you can enter the label `linux` to start the Linux system.

In addition to simply entering `linux`, you can add boot parameters to control how Linux is started. **Boot parameters** are codes similar to module parameters that instruct the Linux kernel how to operate or how to access parts of the computer system's hardware. (Chapter 3 mentioned boot parameters briefly in relation to helping Linux work with hardware components that could not be recognized.)

Besides helping Linux recognize hardware, boot parameters can be used to activate features of Linux. For example, by entering this command at the LILO boot prompt, you start Linux in a special single-user maintenance mode:

```
linux S
```

The Linux kernel supports dozens of boot parameters. Most are used either to help Linux find hardware or to assist with troubleshooting and system administration work. The BootPrompt-HOWTO document describes all of these boot parameters. You can find this document in the directory `/usr/doc/HOWTO`.

On some systems you will be required to add boot parameters each time you start Linux. For example, on many systems you must add a line specifying the amount of memory on the system so that Linux can recognize and use all available RAM. This command would look like this for a system with 128 MB of RAM:

```
linux mem=128M
```

You can configure the LILO program to use certain boot parameters automatically so that you don't have to enter them each time you start the system. The last section of this chapter describes how to change the configuration of LILO.

Initializing System Services

After the LILO boot manager starts the kernel, the kernel initializes all of the computer's hardware. You see messages scroll by as each piece of hardware is initialized. When the boot process is complete and you have logged in to Linux, you can execute the **`dmesg`** command to view the messages printed by the kernel during the boot process.

After the kernel has completed the hardware initialization, the kernel launches a program called `init`. The **`init` program** is a master control program that starts many other processes on the system, such as the login prompts. The `init` program also runs many scripts that initialize the system services you have configured during the installation of Linux. (A **script** is a collection of commands, similar to a macro, that are stored in a text file and executed without user intervention.)

The `init` program is controlled by the `/etc/inittab` configuration file. This file contains pointers to the scripts that `init` will run to initialize the Linux system each time it is turned on. The `/etc/inittab` file is slightly different in each version of Linux, but after reviewing the file on one system, you should be able to recognize which features work on other Linux systems. Almost all of the files referred to by the `/etc/inittab` configuration file are located in the `/etc/rc.d` subdirectory. The main configuration files referred to by the `/etc/inittab` file in Red Hat Linux are described in the list below. (Each is located in the directory `/etc/rc.d`.)

- `rc.sysinit`: this is the main system initialization script for Red Hat Linux. It includes commands for setting up how the keyboard is used, which environment variables are needed, and many other hardware-specific configuration details that are not handled by the Linux kernel.
- `rc`: this script starts system services such as networking, a Web server, an e-mail server, and many others. The specific services that are started depend on how the system is configured (you may have set this up during the Linux installation). The `rc` script is used to set up runlevel-related services, as described in the next section.
- `rc.local`: this script is executed after other initialization scripts. It is initially empty or almost empty. Rather than containing commands by default, `rc.local` is the script in which you can place commands that you want to have executed each time the system is turned on. Examples include loading modules using the `insmod` or `modprobe` command, or executing a special program that you want to have running all the time.

Although a detailed explanation of the contents of each initialization script is beyond the scope of this book, by knowing where the scripts are located, you can research the initialization process yourself to learn more.

Runlevels

A **runlevel** is a mode of operation that provides a particular set of services. Table 4-5 shows the standard runlevels for most Linux systems, with a brief description of how each is used. (The numbers associated with each runlevel's functionality may vary somewhat from one Linux distribution to another.)

Table 4-5 Runlevels in Linux

Runlevel	Name	Description
0	Halt	Used to shut down all services when the system will not be rebooted.
1	Single-user mode	Used for system maintenance. Does not provide networking capabilities.
2	Multiuser mode without networking enabled	Rarely used except for system maintenance or testing.
3	Regular multiuser networking mode	Most systems start in this runlevel.
4		Not used.
5		Identical to runlevel 3, except a graphical login screen is provided; the system remains in graphical mode at all times.
6	Reboot	Used to shut down all services when the system will be rebooted.

The runlevel used when starting the Linux system is defined in the `/etc/inittab` file. The `init` program launches the `rc` script located in the `/etc/rc.d` directory with a parameter that includes the runlevel to use. The `rc` script then starts the appropriate system services based on the selected runlevel.

Each runlevel is associated with a subdirectory. These subdirectories are located in the `/etc/rc.d` directory. For example, the directories `/etc/rc.d/rc3.d` and `/etc/rc.d/rc6.d` include files that control which system services are used in runlevel 3 or 6, respectively.

The runlevel subdirectories, such as `/etc/rc.d/rc3.d`, contain files that indicate which services are to be started or stopped when using that runlevel. Each file in these directories begins with a `K` or an `S`, followed by a two-digit number. The number indicates the order in which services are started or stopped. Services that begin with a `K` are stopped (*killed*); services that begin with an `S` are started. For example, the contents of a typical `/etc/rc.d/rc3.d` directory are shown here:

```
K15gpm          S11portmap      S60nfs
K20rstatd       S15netfs        S75keytable
K20rusersd      S20random       S80sendmail
K20rwhod        S30syslog       S85httpd
K45named        S40atd          S85sound
```

```
K55routed           S40crond            S90xfs
K60mars-nwe         S45pcmcia           S91smb
K92apmd             S50inet             S99linuxconf
S10network          S60lpd              S99local
```

Here you can see that as Linux begins to use runlevel 3, the `rc` script will start the network services, the HTTP server, the Samba server, and many other services that are marked with an `S`. The contents of the `/etc/rc.d/rc6.d` directory shown below indicate a similar list of services that will be stopped when the system is being rebooted. Notice that the numbers in this case are in approximately the reverse order. The first services (network access, for example) form a foundation for the entire system and are therefore the last services to be stopped.

```
K00linuxconf        K20rwhod            K80random
K05keytable         K30sendmail         K85netfs
K10xfs              K45named            K89portmap
K15gpm              K50inet             K90killall
K15httpd            K55routed           K90network
K15sound            K60atd              K92apmd
K20nfs              K60crond            K96pcmcia
K20rstatd           K60lpd              K99syslog
K20rusersd          K60mars-nwe         S00reboot
```

The initialization of each of these services includes another level of complication. The files you see in the runlevel subdirectories (such as those just shown) are not regular files—they are pointers to scripts that stop and start the services. Looking at a single example of this should clarify the point. If you use the `ls -l` command to see a long listing of the HTTP service in the `/etc/rc.d/rc3.d` subdirectory, you see that the `S85httpd` file is actually a pointer to another file: `/etc/rc.d/init.d/httpd`. Because the file (the pointer) in the `/etc/rc.d/rc3.d` subdirectory contains a leading `S`, the `rc` script executes this script with the word `start` after it. Thus, the `rc` script is actually executing this command to start the HTTP service:

```
/etc/rc.d/init.d/httpd start
```

The pointers described here are called symbolic links. A symbolic link allows one file to refer to another file on the system. You'll learn much more about symbolic links in Chapter 6.

Figure 4-10 shows how the initialization components described in this section relate to each other. Each file in the various runlevel directories operates in the same manner as the `S85httpd` pointer just described. All of them point to scripts stored in the `/etc/rc.d/init.d` directory. The scripts in this directory provide an organized method of starting and stopping system services. Although most of the services listed pertain to networking, which is not discussed in detail in this book, you need to understand the concept because these scripts make it easy for you to change almost anything on a Linux system (short of using a new Linux kernel) without restarting the computer. For example, suppose you had reconfigured all of the networking information on a Linux system. Rather than restart the system, you could simply execute these two commands to reinitialize networking:

```
/etc/rc.d/init.d/network stop
/etc/rc.d/init.d/network start
```

Some systems support the use of this single command:

```
/etc/rc.d/init.d/network restart
```

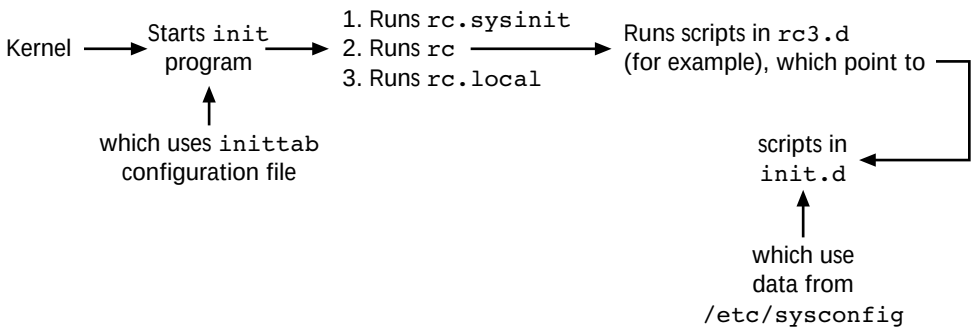

Figure 4-10 The Linux initialization components

Different Linux systems provide various utilities to manage which services are started using this runlevel directory system. One example is the `ksysv` program for KDE, shown in Figure 4-11. Using this utility, you can drag and drop icons representing services to determine which services are started or stopped in which runlevels. The `ksysv` program is not included with all versions of the KDE Desktop; you can look for it on the `System` menu in KDE. The `LinuxConf` utility in Red Hat Linux also provides a capability similar to `ksysv`.

The name `ksysv` may seem odd. The initial `k` simply means that the program is designed to run on the KDE Desktop. The reason for the `sysv` designation is historical. The initialization system just described for Linux is the same basic system that has been used for a long time in UNIX System V (a major version of the UNIX operating system). That system was adopted for Linux, and experienced system administrators know it as a standard System V (pronounced "system-five") initialization process—thus the name `ksysv`, for a KDE utility used to configure *system 5*-style initialization.

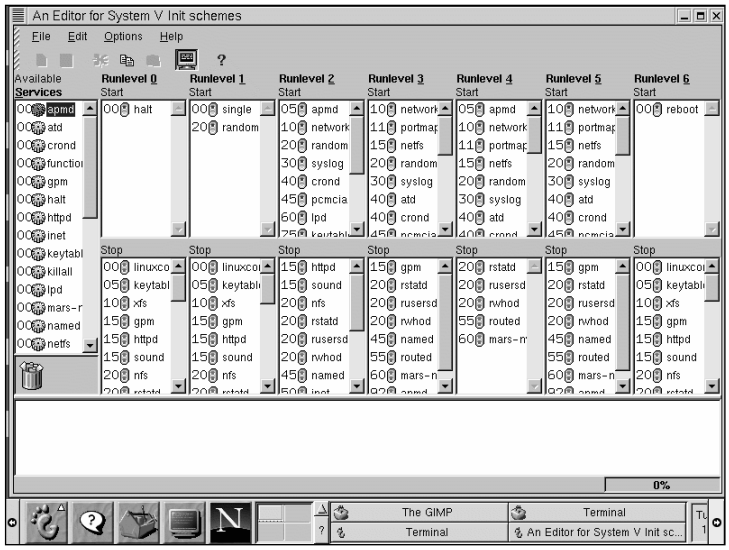

Figure 4-11 The `ksysv` utility

The initialization scripts located in `/etc/rc.d/init.d` are provided for you when you install Linux. In addition, if you install a new software package using the `rpm` command, a script will be placed in the correct directory if the package you install requires one to start a service. Of course, relatively few software packages are used for system services such as a Web server or e-mail server. The point is simply that you don't need to prepare these scripts yourself.

The initialization scripts in `/etc/rc.d/init.d` usually rely on a set of configuration information located in the `/etc/sysconfig` directory (and its subdirectories). Each file in `/etc/sysconfig` is named for a service, and each file contains name-value pairs that define for the initialization script how the service should be configured. For example, the `/etc/sysconfig/network` file on Red Hat Linux looks like this:

```
NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME="incline"
```

These lines are used by the script `/etc/rc.d/init.d/network` to control how networking is set up. (Specific network information is located in the `network-scripts` subdirectory of `/etc/sysconfig`, but a further explanation of networking is beyond the scope of this book.)

Although you can edit the files in `/etc/sysconfig` directly, it is often best to try using the standard configuration tool provided with your Linux distribution. For example, use `COAS` on OpenLinux, `YAST` on Suse Linux, `LinuxConf` on Red Hat Linux, and so forth. Consult your Linux system documentation if you have questions about which utility to use.

Shutting Down Linux

In addition to being familiar with the initialization process of Linux, you need to know how to properly shut down a Linux system. The most important reason for properly shutting down a Linux system is because Linux caches hard disk data in memory. All modern operating systems use caching to improve system performance. **Caching** is the process of storing data from the hard disk in RAM so that it can be accessed more rapidly (because RAM is much faster than a hard disk). But caching data in RAM instead of writing it immediately to the hard disk entails a risk: if you turn off the computer suddenly, data in RAM may never be written to the hard disk and will then be lost.

For this reason, it is important to shut down Linux in an orderly, or graceful, way. The term **graceful shutdown** means stopping all Linux services and shutting down all file access in an orderly way before turning off or rebooting the computer. You can use the following methods for performing a graceful shutdown:

- Enter the command **`reboot`**. This will shut down all services and then restart the computer.
- Enter the command **`halt`**. This will shut down all services and then stop the computer with the message "System halted." This message indicates that you can safely turn off the computer.
- Use the **`shutdown`** command with a parameter to indicate how long to wait before shutting down the system and a parameter to indicate whether the system should be rebooted or halted. For example, to halt the system, beginning in five minutes, use the command `shutdown -h 5`.
- Press Ctrl+Alt+Del. This executes a shutdown command immediately. (The command the system will execute when you press Ctrl+Alt+Del is configured in the `/etc/inittab` file.)
- Enter the command `telinit 0` to halt the system, or the command `telinit 6` to reboot the system (shutting down all services first in both cases). The **`telinit`** command switches the system to a different runlevel: 0 or 6 in this case.

If you are working in a graphical desktop, you should log out before shutting down the system. This allows your graphical setup to be stored for use the next time you start the desktop. After logging out, you will either return to a character-mode prompt (if you are in runlevel 3) or a graphical login screen (if you are in runlevel 5). The graphical login screen usually includes a button that you can select to shut down the Linux system.

Although it's important to know how to shut down Linux, many Linux systems are left running for weeks or months (or years) between reboots. Unless you are working on a machine in a computer lab, or you need to change to a new kernel, or install new hardware, you can leave the system running a very long time without any fear of the system crashing or requiring a reboot.

Only the system administrator should be allowed to shut down a Linux system. By using boot parameters or starting the system from a floppy disk, a user could disrupt the system's security by rebooting. Thus, a system administrator should watch for evidence of unauthorized system

reboots to be certain that nothing improper has been done to the system. Chapter 11 describes how to read system log files, which always include information about when the system was rebooted.

Configuring LILO

As you installed Linux you learned about setting up the LILO boot manager in order to launch Linux and, if necessary, other operating systems located on the same computer. In this section you learn how to update the configuration of LILO after the installation is completed. As with other system administration tasks, you must be logged in as `root` to update the LILO configuration.

The configuration of LILO is stored in the file `/etc/lilo.conf`. This file is created during the installation process, but you can use a standard text editor to alter the file as needed at any time. The `lilo.conf` file contains global configuration options as well as a configuration for one or more operating systems. A sample `lilo.conf` configuration file is shown here and described in detail in the paragraphs that follow:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50

image=/boot/vmlinuz-2.2.5-15
      label=linux
      root=/dev/hda3
      read-only
other=/dev/hda1
      label=dos
      table=/dev/hda
```

The `boot=` line identifies where the LILO boot manager should be installed. If this line indicates a partition name, such as `/dev/hda1`, then LILO is stored on the boot sector of that partition. If a hard disk device is named, without a partition number, such as `/dev/hda`, then LILO is stored on the Master Boot Record of that hard disk. The next few lines in `lilo.conf` define the boot manager program that will be stored at the indicated location and, possibly, a message file to print as LILO starts.

The `prompt` keyword is normally included on systems that support more than one operating system. This allows you to choose which operating system to launch. The `timeout` field provides a waiting time in increments of 1/10 of a second. For example, the line `timeout=50` causes LILO to pause for five seconds before starting the default operating system.

The first few lines in `lilo.conf` (through the `timeout=50` line) are called the global section. These lines define how LILO behaves overall. After the global section, additional sections are defined for each of the operating systems that LILO can launch. In the output above, two types of systems are defined. In order for LILO to boot a Linux operating system you must use the keyword `image` followed by the name of the kernel image file (such as `vmlinuz` or

`zImage`). Additional options below the `image` line define how Linux is loaded. The configuration options for a non-Linux operating system are indicated by the keyword `other`. The existence of an `other` section within this file indicates that LILO should pass control to another partition. For example, if you want LILO to start a Windows system, the `other` section should indicate the partition number where Windows is stored, like this:

```
other = /dev/hda1
```

The `other` partition must have some method of starting the operating system located on that partition. (Windows partitions are able to start Windows by default.)

For both Linux and an `other` operating system, you can specify additional parameters. You can only use a few of these parameters in the `other` sections. One that you can use is `password`, which requires the user to enter a password before LILO will cede control to the other operating system. The `password` parameter looks like this:

```
password=mypassword
```

The `image` sections (used to start a Linux operating system) support many options. One of these options is `append`, which is used to add boot parameters to Linux:

```
append="mem=128M"
```

The line beginning with `label` names the `image` section or the `other` section within the `lilo.conf` file. The values of all the `label` fields are displayed when you boot the system and press Tab to display operating systems that LILO can start.

You can review the documentation for the `lilo.conf` configuration file to discover all of the possible options that apply to the global section (the first part of the file) and to the `image` and `other` sections. To view the documentation, enter the command `man lilo.conf`.

Because LILO loads the Linux kernel as a file when the system is started, LILO can choose among several Linux kernels. When you recompile the kernel as described previously in this chapter, you can update `lilo.conf` so that LILO can start either your previous (existing) kernel or the new kernel that you just created. It is wise to provide configuration options for both in case the new kernel that you created has an unexpected problem—you can still reboot the system and have LILO start the old kernel that works. The following lines show how the two `image` sections of `lilo.conf` would look if you wanted to configure LILO so that it could start one of two different Linux kernels stored on the hard disk.

```
image=/boot/new_vmlinuz
      label=new_kernel
      root=/dev/hda4
      read-only

image=/boot/vmlinuz-2.2.5-15
      label=linux
      root=/dev/hda4
      read-only
```

After you have set up the `lilo.conf` file to support all of the operating systems and options needed, you must update the information on the hard disk. Saving the `lilo.conf` file records the configuration data, but it does not activate it on the boot sector or Master Boot

Record. To do that, you must run the `lilo` command. The **`lilo`** command reads the `lilo.conf` configuration file and updates the hard disk boot information based on the configuration you set up. Using this command is simple:

```
/sbin/lilo
```

If you update the `lilo.conf` file but do not run `lilo`, the configuration of LILO remains unchanged when you reboot the system. When `lilo` runs successfully, it displays the name of each operating system label for which you have defined a section.

In rare instances, you will only see part of the word `LILO` when you boot the system (for example, you may see just `LI`), and the system then stops responding without booting an operating system. This indicates that LILO cannot locate information to start an operating system. In this case you must start the system from a boot disk or rescue disk and change the `lilo.conf` file. See Chapter 9 for more information on using these disks.

CHAPTER SUMMARY

- All files in Linux are stored in a single directory structure beginning with `/`, the root directory. You can use standard commands as well as graphical tools to move around the directory structure and manage files. All Linux files and directories have file permissions assigned based on the user assigned as owner and an assigned group name.
- Software packaging schemes, especially rpm, are used to install and manage applications on a Linux system. Graphical and command-line utilities are available to manage software packages. The `tar` command—one of many ways that files can be compressed and combined in Linux—is also popular on Linux systems.
- The Linux kernel uses modules to extend its functionality. Linux version numbers are significant in denoting the status of a particular version of the kernel. The kernel can be recompiled from source code after setting up a configuration for the new kernel using a menu-based tool. A new kernel must be configured in LILO so that it can be launched.
- Linux uses a standard System V–style initialization process in which the `init` program starts many configuration scripts to configure the system and launch system services. Using a complex system of subdirectories, the `rc` script and the default runlevel define which services are started. Linux must be properly shut down. The LILO boot manager can be configured with many different options to manage how Linux is started.

KEY TERMS

boot parameters — Codes similar to module parameters that are used to instruct the Linux kernel how to operate or how to access parts of the computer system's hardware.

caching — The process of storing data from the hard disk in RAM so that it can be accessed more rapidly (because RAM is much faster than a hard disk).

`cat` — Command used to print the contents of a file to the screen.

`cd` — Command used to change the directory you are working in (the current working directory).

`chmod` — Command used to change the file permissions assigned to a file or directory.

`chown` — Command used to change the ownership of a file or directory.

command-line option — A command-line parameter.

command-line parameter — An additional piece of information (besides the program name) that is included on the command line when a program is started.

command-line window — A window within a graphical environment that permits you to enter commands at the keyboard.

`cp` — Command used to copy a file or directory from one location or name to another.

current working directory — The directory in which you are working.

`dmesg` — Command used to view the messages printed by the kernel during the boot process.

`echo` — Command used to print text to the screen, converting variable names to their corresponding values.

environment variables — Variables that are defined by the Linux shell so that all programs can access their values.

execute permission — A file permission that allows a user to launch a file as a program or see a file within a directory. Represented by a letter `x`.

`file` — Command used to print a summary of the type of data contained in a file.

file extension — The last part of a filename after a period.

file manager window — A graphical window that displays the contents of a directory (usually as a collection of icons) and lets you work with the files and directories using menus, mouse clicks, and dialog boxes.

file permissions — Codes that define the type of access that a user has to a file or directory on the Linux system.

function library — A file containing a collection of commonly used functions that any program can use as it runs.

graceful shutdown — The technique of stopping all Linux services and shutting down all file access in an orderly way before turning off or rebooting the computer.

group — A named account that consists of a collection of users. Each member of a group has access to files owned by that group.

group permissions — A set of three file permissions (`r`, `w`, and `x`) that apply to members of the group assigned to a file or directory.

`gunzip` — Command used to uncompress a file that has been compressed using `gzip`.

`gzip` — Command used to compress any file on a Linux system.

`halt` — Command used to shut down all services and then stop the computer with the message "System halted."

home directory — The location where all of a user's personal files are stored.

`init` — Command used to switch the system to a different runlevel.

`init` **program** — A master control program that starts many other processes on the system, such as the login prompts.

`insmod` — Command used to copy a module file from the hard disk and add it to the Linux kernel running in memory.

kernel modules — Files containing computer code that can be loaded into the kernel or removed from the kernel as needed.

`ldd` — Command used to list the function libraries that a program uses.

`less` — Command used to print the contents of a file one screenful at a time. It allows you to move around in the file and otherwise control the command by using the keyboard.

`lilo` — Command used to read the `lilo.conf` configuration file and update the hard disk boot information based on the configuration.

`ls` — Command used to list the files in a directory.

`lsmod` — Command used to list the modules that are installed in the Linux kernel.

`mkdir` — Command used to create a new directory.

`modprobe` — Command used to load a module with all of its required supporting modules.

module parameters — Information needed by a module to locate system resources. The parameters are added after the module name when using the `insmod` or `modprobe` command.

`more` — Command used to print the contents of a file one screenful at a time. The `more` command is similar to the `less` command but with fewer keyboard control options.

`mv` — Command used to rename a file or move it to a new location.

other permissions — A set of three file permissions (`r`, `w`, and `x`) that apply to all users on the Linux system who are not the owner of the file or directory in question and are not members of the group assigned to the file or directory.

parent directory — The directory that is one level above the current directory.

`pwd` — Command that displays the current working directory.

read permission — A file permission that allows a user to read the contents of a file or browse the files in a directory. Represented by a letter `r`.

`reboot` — Command used to shut down all services and then restart the computer.

release number — A number assigned by the company that prepares a Linux product. It allows the company to track how many times the kernel file has been altered before the final product is shipped.

`rm` — Command used to delete a file.

`rmdir` — Command used to remove (delete) an empty directory.

`rmmod` — Command used to remove a module from the kernel.

root directory — The starting point for all access to Linux resources. It is indicated by a single forward slash: `/`.

`rpm` — Command used to manage all of the rpm software packages on a Linux system.

runlevel — A mode of operation that defines which Linux system services are started and which are shut down. The standard Linux runlevel of 3 includes networking and other common services such as system logging and task scheduling.

script — A collection of commands, similar to a macro, that are stored in a text file and executed without user intervention.

shell — The program in Linux that captures and handles commands entered in a command-line environment.

`shutdown` — Command used to shut down Linux gracefully.

`tar` — Command used to create a single file that contains many other files, often compressed to save space.

tar archive — A file created by the `tar` command.

`telinit` — Command used to switch the system to a different runlevel.

terminal emulator window — A command-line window (also called a terminal window) within a graphical environment.

timestamp — A record of the date and time when an event occurred.

`touch` — Command used to create a new file with no data in it or update the access timestamp of an existing file.

`umask` — Command used to set the file permissions assigned when you create a new file.

`uname` — Command used to provide information about the operating system, including the kernel version.

user permissions — A set of three file permissions (`r`, `w`, and `x`) that apply to the owner of a file or directory.

variable — A memory location used by a program to store a value, such as a number or a word. Each variable is assigned a name so that the program can access the value by referring to the name.

write permission — A file permission that allows a user to add or change information in a file or create files within a directory. Represented by a letter `w`.

`xterm` — A program within a graphical environment that provides a command-line window.

`zcat` — Command used to print the contents of a compressed file to the screen.

REVIEW QUESTIONS

1. The _____ directory is the beginning of the Linux directory structure.
   a. `/`
   b. `/root`
   c. `/home`
   d. `/dev/hda`
2. The `pwd` command is used to:
   a. Process writeable domains on the network
   b. Control power used by a Linux system
   c. Print the current working directory to the screen
   d. Print a summary of the writeable devices on the system
3. As with a DOS system, the command `cd..` can be used to change to the parent directory at any time. True or False?
4. Name three commands that can be used to view the contents of text files.

5. The command `Ls -l` is invalid because:
   a. Linux commands cannot contain hyphens.
   b. Linux commands are case sensitive.
   c. The `ls` command is not a real command.
   d. The `-l` option is not supported by the `ls` command.
6. The command `chmod 652 test` grants _____ execute permission to the `test` file.
   a. user
   b. group
   c. other
   d. all users on the system
7. The owner and group assigned to a file are shown by which of the following commands?
   a. `chown`
   b. `ls -l`
   c. `modprobe`
   d. `useradd`
8. Execute permission on a file is required to
   a. Launch that file as a program
   b. Create a directory with a matching name
   c. Allow other users to read the file
   d. Use the `insmod` command to add the file as a module
9. Describe the purpose of function libraries and name two directories where they are commonly located.
10. The command `rpm -q packagename` does the following:
    a. Determines whether `packagename` is installed on the system
    b. Locates `packagename` on a CD-ROM
    c. Erases `packagename` if it is currently installed
    d. Summarizes the disk quota for users of `packagename`
11. The `tar` command creates archive files that are commonly compressed by the _____ command.
    a. `bzip`
    b. `compress`
    c. `zip`
    d. `gzip`
12. Contrast the advantages of `rpm` and `tar` formats.

13. Which of the following represents a development release of the Linux kernel?
    a. 1.2.13
    b. 2.1.42
    c. 2.0.10
    d. 2.2.5
14. Name two methods of determining the version and timestamp of the kernel currently running on a Linux system.
15. The module files loaded by the `insmod` command are located in a subdirectory of `/etc/modules`. True or False?
16. Red Hat Linux uses a program called _____ to automate loading of kernel modules as they are needed.
    a. `modprobe`
    b. `kerneld`
    c. `find`
    d. `/lib/modules`
17. Name four commands used to work with kernel modules.
18. The terms *module parameter* and *boot parameter* refer to the same basic thing. True or False?
19. The process of recompiling the Linux kernel does *not* include:
    a. Creating a configuration file
    b. Making certain the kernel source code is installed
    c. Running the `make` command
    d. Inserting kernel modules that provide support for recompiling the kernel
20. After creating a new kernel file you must reconfigure LILO so that:
    a. The new kernel can be started by the LILO program when the system is rebooted
    b. The older kernel version is gracefully unloaded from memory
    c. Existing kernel modules do not cause a memory conflict when loaded
    d. Security enhancements remain in force
21. The _____ program displays kernel hardware configuration messages from the system boot process.
    a. `init`
    b. `chmod`
    c. `dmesg`
    d. `uname`
22. Why is `modprobe` preferred over `insmod` as a tool for adding modules to the kernel?

23. The `init` program uses the following configuration file:
    a. `/etc/lilo.conf`
    b. `/etc/inittab`
    c. `/etc/modules/inittab`
    d. `/etc/rc.d/rc`
24. Name the two runlevels normally used to run a Linux-based computer, and describe the difference between those two runlevels.
25. Files in the runlevel subdirectories, such as `/etc/rc.d/rc3.d`, are actually pointers to scripts in `/etc/rc.d/init.d`. True or False?
26. The files in `/etc/rc.d/init.d` can be used to:
    a. Automatically insert kernel modules
    b. Stop and restart most standard services in Linux
    c. Reconfigure the LILO program after recompiling the kernel
    d. Set default file permissions for the `root` user
27. The scripts in `/etc/rc.d/init.d` are provided by:
    a. The system administrator who installs Linux
    b. The `rc` script, which runs before any of the `init.d` scripts
    c. The rpm that installs the service that the script controls
    d. The kernel itself
28. The configuration data stored in files within the `/etc/sysconfig` directory can be modified using many different configuration utilities. True or False?
29. Name three commands that can be used to begin a graceful shutdown of Linux.
30. Describe why the LILO program must be executed after making changes to the `lilo.conf` file.

HANDS-ON PROJECTS

Project 4-1

In this activity you practice using Linux commands for managing files and directories. To complete this activity you should have available an installed Linux system on which you have a valid user account (one for which you know the password).

1. Log in to Linux using your user account name and password.
2. If you logged in using a graphical login screen, open a terminal window by opening the main menu of your desktop environment, pointing to the **`Utilities`** menu, and then clicking on **`Gnome Terminal`** (in the Gnome Desktop) or **`Terminal`** (in the KDE Desktop).
3. Enter the **`pwd`** command to display your current working directory. Because you have just logged in, this should display your home directory.

4. Create a new subdirectory within your home directory using the command **mkdir archive**.
5. Change to the new subdirectory you just created by entering **cd archive**.
6. Create a new file named `report` using this `touch` command: **touch report**.
7. Enter **ls -l** to view a long-format listing of the files in the `archive` subdirectory. Can you identify your username in the command output as the owner of the file `report`?
8. Notice the file permissions in the output of the `ls -l` command from the previous step. What default file permissions were assigned? Use this `chmod` command to remove all permissions for the group and other users: **chmod go-rwx report**. How could you reset the file permissions using a three-digit code in the `chmod` command?
9. Enter **ls -l** again to review the file permissions with your changes.
10. Change the name of the `report` file to `oldreport` using this `mv` command: **mv report oldreport**.
11. Use the `ls` command to verify that the file `report` is no longer there—it has been replaced by `oldreport`.
12. Copy a system file from the `/etc` directory by entering: **cp /etc/termcap .** (Remember to include the period to indicate that you want to copy the file to your current directory.) Do you see any feedback when a command is successful?
13. Use the `cat` command to view the contents of the file that you copied in the previous step: **cat termcap**. Can you read the contents?
14. Use the `less` command to view the contents of the file: **less termcap**. Use the Page Up and Page Down keys to scroll through the file. Press **q** to exit the `less` command.
15. Enter the **ls -l** command. Note the size of the `termcap` file.
16. Compress the `termcap` file using the `gzip` command: **gzip termcap**.
17. Use the `ls -l` command to view the directory contents. How has the filename changed? How has the size of the file changed?
18. Use the `file` command to display the type of data contained in the `termcap.gz` file: **file termcap.gz**.
19. Erase the `termcap.gz` file using the `rm` command: **rm termcap.gz**.
20. Change back to your home directory by entering **cd ..** (don't forget the space before the two periods).
21. Erase the `archive` directory using the `rmdir` command: **rmdir archive**. Why won't the command work? How could you make it work?

Project 4-2

In this activity you use several commands to learn about the Linux kernel running on a computer. To complete this activity you should have available an installed Linux system on which you have a valid user account (one for which you know the password).

1. Log in to Linux using your user account name and password.

2. If you logged in using a graphical login screen, open a terminal window by opening the main menu of your desktop environment, pointing to the **Utilities** menu, and then clicking on **Gnome Terminal** (in the Gnome Desktop) or **Terminal** (in the KDE Desktop).
3. Enter the command **uname -r** to see which version of the Linux kernel is running on the computer. Do you see a release number in the output?
4. Enter the command **uname -v** to see the timestamp of the Linux kernel.
5. Enter the command **cat /proc/version**. How does the information displayed compare to the output of the `uname` command?
6. Review the contents of the `lilo.conf` file by entering the command **less /etc/lilo.conf**.
7. Locate the section of the `lilo.conf` file beginning with the line `image=`, and note the directory and filename listed.
8. Press **q** to exit the `less` command.
9. Change to the directory indicated on the `image=` line in `/etc/lilo.conf`: **cd /boot**.
10. Enter the **ls -l** command to see the files in the `boot` directory. Can you locate the file named in the `image` section of `lilo.conf`? That file is the Linux kernel. How large is the file?
11. Enter the command **dmesg | less** to review the kernel boot messages. What parts of the system hardware do you recognize in the output? How might this output help you manage the system's hardware?

Project 4-3

In this activity you review the initialization information on the Linux system. To complete this activity you should have available an installed Linux system on which you have a valid user account (one for which you know the password).

1. Log in to Linux using your user account name and password (not the `root` account).
2. If you logged in using a graphical login screen, open a terminal window by opening the main menu of your desktop environment, pointing to the **Utilities** menu, and then clicking on **Gnome Terminal** (in the Gnome Desktop) or **Terminal** (in the KDE Desktop).
3. Enter the command **cat /etc/sysconfig/network**. What configuration options do you recognize in the output?
4. Change to the `rc.d` initialization directory: **cd /etc/rc.d**.
5. Enter the **ls** command. What filenames do you recognize from the discussion in the chapter?
6. List the files in the `init.d` subdirectory: **ls init.d**. Can you recognize any network or other services that match the names of the files in this directory?
7. List the files in the `rc3.d` subdirectory: **ls rc3.d**. How do the files in this directory correspond to those in the `init.d` subdirectory?

8. Change to the `init.d` subdirectory: **`cd init.d`**.
9. View the `syslog` script using the `ls -l` command: **`ls -l syslog`**. Notice the file permissions assigned to the script. Who is permitted to read the script? (The `syslog` script controls the system logging programs described in Chapter 11.)
10. Execute the `syslog` script using the command **`./syslog restart`** (don't forget the **`./`** at the beginning to indicate that the file is located in the current directory). Wait a few moments for all of the messages to appear on screen and the command prompt to return. What can you conclude about the file permissions allowing everyone to read the script?
11. Use the `less` command to look at the contents of the script: **`less syslog`**. You can learn about creating scripts by reviewing existing scripts on the system. (Chapter 12 describes how to create your own scripts.) Press **`q`** to exit the `less` command.
12. Change back to your home directory using the command **`cd`** with no parameters. (This always returns you to your home directory.)
13. Enter **`pwd`** to verify that you are in fact in your home directory.

CASE PROJECTS

1. You are helping a local company set up a research lab containing several Linux-based computer systems that will be used for various scientific experiments. The experiments require high performance from Linux. Each computer system includes a fast hard disk attached to a SCSI controller, a high-speed networking card, and 512 MB of RAM. As you install Linux on the system, you notice that the SCSI card is not supported by the default kernel. A kernel module is required to support the SCSI cards in the lab. What problems might come up if Linux were installed on a SCSI hard disk, but the SCSI hard disk could not be accessed without first loading a module into the kernel?
2. Given what you know about the Linux kernel, does it seem possible that you could create a software package containing a new Linux kernel that included built-in support for the lab's SCSI device, and then add that package to each system in the lab? Do you think this would be worth the time required to prepare such a package?
3. Each computer in the lab is slightly different, though all have the same type of SCSI controller card installed. After installing Linux on each computer, one of the computers will not boot to Linux. What could you check to identify differences in the computer's hardware or configuration? Could using a boot parameter help? Could using a boot disk help?